Laboratorio Nro. 5: Programación Dinámica

Manuela Valencia Toro
Universidad Eafit
Medellín, Colombia
mvalenciat@eafit.edu.co

Felipe Olaya Ospina
Universidad Eafit
Medellín, Colombia
folayao@eafit.edu.co

Laura Sánchez Córdoba
Universidad Eafit
Medellín, Colombia
lsanchezc@eafit.edu.co

3) Simulacro de preguntas de sustentación de Proyectos

1. Se utilizan múltiples estructuras para el desarrollo del algoritmo, siendo las principales mapas, sets, listas y un grafo, para el almacenamiento y obtención de los datos para la solución del problema del Agente Viajero.
   En primer lugar, se tiene un mapa para almacenar los subconjuntos y el costo de cada uno en su viaje desde la posición inicial, y otro para obtener el vértice del que parte cada recorrido por los subconjuntos (cabe recalcar que se tiene una clase Index la cual es la clave de los mapas y que actúa como indicador del vértice y los subconjuntos existentes). Se genera una lista de sets en la que cada uno es un subconjunto del conjunto total de vértices en el grafo, se recorre cada vértice y se crea con cada uno de los set un Index para el cual buscaremos mediante un ciclo cuál tendrá menor costo, se halla y se almacena dicho costo y el vértice del cual se partió (este costo se halla a partir de los pesos almacenados en los arcos entre dos vértices), se suma cada costo que se obtenga al recorrer cada vértice (por subconjuntos) y regresar al inicio y se retorna el valor de éste.
   Tomado de: https://github.com/mission-peace/interview/blob/master/src/com/interview/graph/TravelingSalesmanHeldKarp.java

2. Diversas versiones de los algoritmos genéticos han sido presentadas por investigadores, con la intención de mejorar su eficiencia en la solución del Problema del Agente Viajero. El esquema de codificación elegido es la permutación, en donde los genes dentro de los cromosomas representan el orden en el cual uno tiene que viajar desde la ciudad inicial. La idea aquí es que, si existen n ciudades en el problema elegido, estos se han dividió en n/x subgrupos donde x representa el número de cromosomas en toda la población, y n representa el tamaño de los clústeres. El propósito del enfoque efectivamente busca localmente y combina el mejor resultado local desde una solución global. Todas las ciudades del problema son agrupadas en subgrupos, usando K-subgrupos, donde cada grupo tendrá una colección de ciudades (Figura 1). La selección de los padres, operadores como cruza o mutación son aplicados y el mejor ordenamiento de las ciudades dentro de cada grupo es calculado independientemente como se muestra en la figura 2 y finalmente los resultados desde cada grupo son combinados para formar una simple solución como se muestra en la figura número 3.

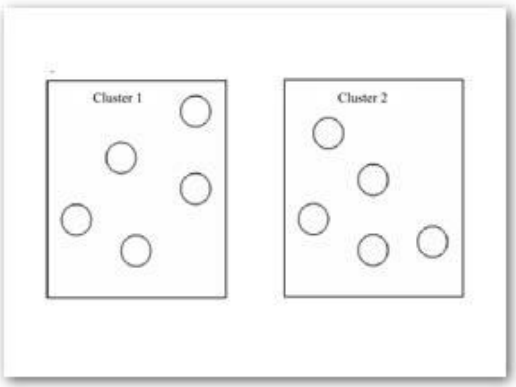

Figura 1. Agrupación de Cromosomas en K subgrupos.

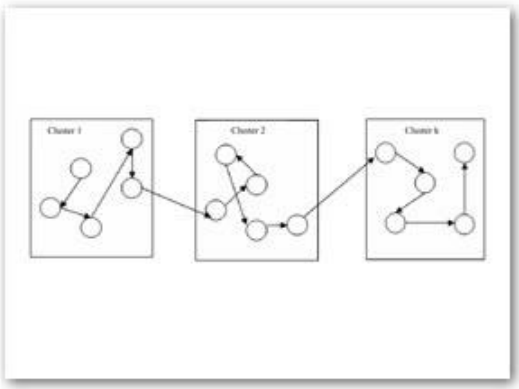

Figura 2. Mejor solución local de cada subgrupo.

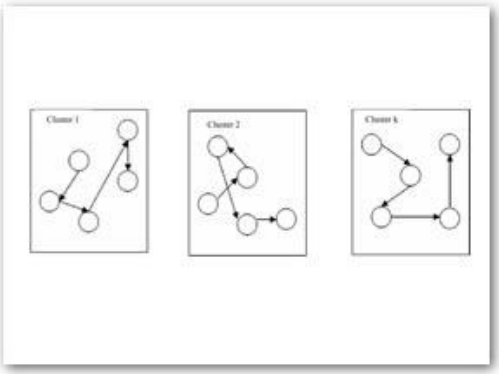

Figura 3. Solución final global para cada problema.

Se tienen otras variantes de solución como la fuerza bruta (la cual no es nada eficiente) pues prueba todas las combinaciones posibles y la Programación dinámica la cual se trabaja a fondo en este laboratorio, la cual, a partir del almacenamiento de los datos hallados por el mismo algoritmo, hace la eficiencia mucho mejor.
Tomado de: https://repository.uaeh.edu.mx/revistas/index.php/icbi/article/download/485/479?inline=1

3. El algoritmo usa además de las ya mencionadas en el numeral 1.2 (dado que se trabaja con el algoritmo Held-Karp) estructuras de datos como listas y arreglos para el manejo y almacenamiento de los datos, los cuales se leen desde la consola con un BufferedReader y a partir de estos se genera un grafo bajo la representación de listas de adyacencia, pues no se tiene garantía de que exista una cantidad de arcos muy grande, por lo que a pesar de no ser tan eficiente como una matriz para las búsquedas, nos garantiza menos gasto de espacio en caso de que pocos arcos existan; sin embargo el algoritmo funciona para ambos tipos de grafos.
Se leen y almacenan cada una de las posiciones a las que el robot debe ir y se halla su respectiva distancia (teniendo en cuenta que solo se puede viajar en línea recta), se agregan al grafo y éste se genera, finalmente usando el algoritmo heldKarp de la clase HeldKarp, que recibe dicho grafo, se obtiene la distancia más corta y se imprime.

4.

```
public static void main (String [] args) throws IOException {
```

```
        BufferedReader lector = new BufferedReader (new InputStreamReader(System.in));
        int esc=Integer.parseInt(lector.readLine());

        for(int i=0;i<esc;i++){ //n
           String stam=lector.readLine(); //c
           String[] stamAr= stam.split(" "); //c
           //tamaño del escenario
           int tam=Integer.parseInt(stamAr[0]); //c

           ArrayList<Pair> coord=new ArrayList<Pair>(tam); //c
           //coordenadas (inicial y posteriormente, desechos)
           Pair pos1; //c
           String spos; //c
           spos=lector.readLine();
           String[] posAr=spos.split(" "); //c
           pos1=Pair.makePair(Integer.parseInt(posAr[0]),Integer.parseInt(posAr[1]));  //c
           coord.add(pos1); //c
           int ncoord=Integer.parseInt(lector.readLine()); //c
           for(int j=0;j<ncoord;j++){ //m
              String lineaAc=lector.readLine(); //c'
              String[] lineaAcAr=lineaAc.split(" "); //c'
              Pair pos=Pair.makePair(Integer.parseInt(lineaAcAr[0]),Integer.parseInt(lineaAcAr[1])); //c'
              coord.add(pos); //c'
           }
           Digraph g=new DigraphAL(coord.size()); //c'
           for(int k=0; k<coord.size();k++){ //m
              for(int l=0; l<coord.size();l++){ //m
                 if(k!=l){ //c'
                    int peso=Math.abs((int)coord.get(k).first-(int)coord.get(l).first)
                          + Math.abs((int)coord.get(k).second-(int)coord.get(l).second); //c'
                    g.addArc(k,l,peso); //c
                 }
              }
           }
           int res=HeldKarp.heldKarp(g); //O(2^m * m^2)
           System.out.println("The shortest path has length "+res); //c

        }

     }
}
```

T(n,m)=n*(m+m^2+(2^m * m^2)+c')+ c
T(n,m)=n*(m+m^2+(2^m * m^2)+c') R.S.
T(n,m)=n*(m+m^2+(2^m * m^2)) R.S.
T(n,m)=n*(m^2+(2^m * m^2)) R.S.
T(n,m)=n*(2^m * m^2) R.S.
O(n,m)=n*(2^m * m^2)

5. La variable n hace referencia a la cantidad de escenarios posibles que el usuario puede introducir por la consola y la variable m al número de vértices que hay el en grafo; 2^m hace referencia a la cantidad de

subconjuntos que forman y m^2 hace referencia a que recorremos cada vértice y también buscamos a partir de estos el que es anterior.

4) Simulacro de Parcial

1)

1.1

		C	A	L	L	E
	0	1	2	3	4	5
C	1	0	1	2	3	4
A	2	1	0	1	2	3
S	3	2	1	2	3	4
A	4	3	2	3	4	5

1.2

		M	A	D	R	E
	0	1	2	3	4	5
M	1	0	1	2	3	4
A	2	1	0	1	2	3
M	3	2	1	2	3	4
A	4	3	2	3	4	5

2)

2.1 N*M
2.2 RETURN TABLE[I][J];

3)

3.1 O(n)
3.2 t(n)= c1 :n + c2

4)

Ninguna de las anteriores

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co